COLAP: A Predictive Framework for Service Function Chain Placement in a Multi-cloud Environment

Lav Gupta*
Department of Computer Science & Engineering
Washington Univ. in St. Louis, USA

Mohammad Samaka
Dept. of Computer Science
Qatar University
Qatar

Raj Jain
Department of Computer Science & Engineering
Washington Univ. in St. Louis, USA

Aiman Erbad
Department of Computer Science
Qatar University
Qatar

Deval Bhamare
Department of Computer Science
Qatar University
Qatar

Chris Metz
Cisco Systems
San Jose

* Corresponding author

***Abstract*—Network function virtualization (NFV) over multi-cloud promises network service providers amazing flexibility in service deployment and optimizing cost. Telecommunications applications are, however, sensitive to performance indicators, especially latency, which tend to get degraded by both the virtualization and the multiple cloud requirement for widely distributed coverage. In this work we propose an efficient framework that uses the novel concept of random cloud selection combined with a support vector regression based predictive model for cost optimized latency aware placement (COLAP) of service function chains. Extensive empirical analysis has been carried out with training datasets generated using a queuing-theoretic model. The results show good generalization performance of the predictive algorithm. The proposed framework can place thousands of virtual network functions in less than a minute and has high acceptance ratio.**

Keywords—multi-cloud computing; network function virtualization; service function chain; virtual network function; placement; latency; machine learning; support vector regression

I. INTRODUCTION

Network Function Virtualization (NFV) can potentially provide the benefits of flexible scaling, redundancy and lower total cost of operations. Substantial ground has been covered by European Telecommunications Standards Institute (ETSI) in terms of exposition and group specifications since the release of their first NFV whitepaper in 2012 [1]. Since then there has been increasing interest in the research community on various aspects of NFV. Most carrier networks have wide area and need to use multiple clouds. However, as mentioned in [2], the proposed solutions do not offer real support for some of the core requirements. When it comes to extracting carrier grade performance from NFV, especially in multi-cloud environments, much is still to be done [3], [4]. Two main areas where NFV falls short compared to traditional networks are capacity and performance. In this work we focus on performance, a major issue in telecommunications networks that require control over parameters like latency, jitter and packet loss and uptime of the order of five nines (downtime of just 26 seconds downtime in 30 days). Performance takes a hit when the dynamic telecommunications environment meets the ease of creation, destruction, migration and scaling of NFV as the possibility of uncontrolled virtualization increases. This has led the authors in [5] to comment that virtualization may lead to abnormal latency variations and significant throughput instability irrespective of utilization.

Cloud computing and NFV have a natural synergy and it is expected that industry standard IT cloud technologies, will evolve to support the requirements of telecommunications networks [6]. With multi-cloud infrastructure, the telecommunications service providers (TSPs) can take the advantage of competitive pricing, better points of presence, flexibility of scaling and avoiding single point of failure (In this paper, reference to the term TSP also includes the ISPs, the Internet service providers). In their infrastructure overview ETSI has indicated latency and throughput requirements as the discouraging factors for use of public clouds. Several ITU recommendations, viz., G.107, G.109, G.113, G.114, define standards of various aspects of latency for carrier-grade mobile and fixed telephony.

The Cloud Service Providers (CSPs) have to reconcile the conflicting requirements of high utilization of physical infrastructure with the desired network performance and optimize the cost of eventual placement. The proposed cost optimized latency aware placement (COLAP) framework implements two major concepts: 1) A fast algorithm implementing randomized selection of clouds for optimum cost, and 2) heuristics for placement using predictive containment of latency based on the machine learning technique of support vector regression (SVR).

The rest of the paper is organized as follows. Section II presents a summary of the related work. In Section III, we discuss service chains and their placement in multi-cloud infrastructures. The problem description and solution are in Section IV. In Section V, we present the evaluation results. Finally, Section VI gives summary and ongoing work.

II. RELATED WORK

Much of the work done on virtual machine placement falls into the category of static, reactive and on request from service providers [7]. A lot of work in this category involves setting up

This work has been supported under the grant ID NPRP 6 - 901 - 2 - 370 for the project entitled "Middleware Architecture for Cloud Based Services Using Software Defined Networking (SDN)", which is funded by the Qatar National Research Fund (QNRF) and by Cisco Systems. The statements made herein are solely the responsibility of the authors.

The 7th IEEE Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, Jan 9-11, 2017

the problem as an integer linear program (ILP), mixed integer linear program (MILP), mixed integer quadratic constrained program (MIQCP) or a similar problem with one or more objective optimizing resource level parameters like compute resource usage, storage usage, power consumption along with constraints like capacity and affinity. Since solving the placement problem is NP-hard [8], algorithms like greedy placement and heuristics like first fit decreasing (FFD), have been proposed to limit the time a linear or a quadratic programming solution takes to give a reasonable solution. As far as the VNF placement over NFV Infrastructure (NFVI) is concerned, MILP based solution in [9] takes into account both network level parameters (minimization of link utilization, latency and traffic flow) and NFVI level metric (minimization of computing resources). The authors in [10] have set up the problem as MIQCP to obtain a placement scheme for the network functions and chaining them together considering the limited network resources and functional requirements. Optimization of network operation costs and utilization to determine required number and placement of VNFs has been discussed in [11]. The authors provide dynamic programming based heuristic to solve larger instances with 1.3 times the optimal solution.

More recently dynamic and proactive techniques have been studied. The authors in [12] have proposed an ILP based solution for optimization of the response time. The authors in [13] use machine learning based workload prediction to take care of delays in place and scaling virtual functions. Of the three techniques studied, Support Vector Regression (SVR), Neural Networks (NN) and Linear Programming (LP), SVR displays superior prediction accuracy in a 9-12 minute window. Multi-agent based reinforcement learning approach has been presented in [14]. In [15], the authors apply genetic algorithm to the placement problem to get an algorithm that is faster compared to MILP based solutions and yet provides a good solution. Authors in [16] describe an architecture based on an orchestrator that ensures the automatic placement of the virtual nodes and the allocation of network services on them.

III. SERVICE CHAIN PLACEMENT ON MULTIPLE CLOUDS

This section aims at discussing the terminolgy involved in and the need for dynamic placement of service function chains across mutiple clouds.

A. *Virtual Network Functions and Service Function Chains*

Each network service (NS) is implemented through one or more service function chain (SFC). Each SFC consists of basic services like routing and middle boxes like firewall implemented as virtual network functions (VNF) which are chained to process the traffic in a particular way [17]. A VNF is software based and can be instantiated on virtual machines (VMs) created on commodity servers. Examples are LTE sub-systems like Internet Multimedia Subsystem (IMS) and Packet Gateway (PGW) and network middleware like firewall, load balancers and WAN accelerators. A VNF forwarding graph or an SFC is a set of VNFs or services with a well-defined sequence for the packets to travel.

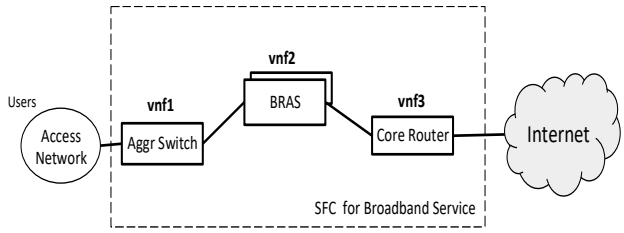

Fig. 1. Broadband service function chain

A *Policy* is a set of rules that define actions to be taken under different conditions. Each policy is implemented as one or more service chains [18]. A simple policy could be to aggregate the subscriber traffic coming through the access network and then route it through the Broadband Remote Access Server (BRAS) for billing and accounting before being sent through the core router to the Internet. In Figure 1, The BRAS is shown to have multiple instances to cater to the amount of traffic requiring processing through this function.

B. *Placing Service Chains Across Multiple Clouds*

Upon request from a Telecommunications Service Provider (TSP), the Cloud Service Provider (CSP) or a cloud broker dimensions the SFC and places the required number of instances of VNFs on available clouds. Some services are dynamic where the type and number of VNFs would change frequently while others may be static where VNFs types may remain the same but capacity requirement may change. In either case the CSP would have to deal with changes in the capacity requirement of VNFs and links between them. If at any time the service level agreement (SLA) conditions are breached, the CSP has to pay the stipulated penalty. Thus, end-to-end latency and processing delays need to be continually monitored and managed whether the placement is static or dynamic.

Placing service chains as a unit rather than individual functions separately yield better results [19]. It gives an opportunity to achieve global minima for the parameter being optimized when placing a full chain. At the infrastructure level inter-VNF communication overheads can be reduced. If sufficient resources are not available to implement full service chains, then the request is rejected or, if the policy permits, degraded service (for instance without a firewall) is provided [17], [20].

A tenant's profile is specified as $<c_N, v_1, v_2 \ldots v_m, p>$ for each request. Here $v_1 \ldots v_m$ gives the types of VNFs and order of traffic traversal (assuming a linear chain), c_N is the native cloud through which the traffic enters and p is the desired packet rate (packet/second). Other stipulations like cost optimization and latency threshold (L_{th}) are part of the SLA. All the requests of the tenant service providers are consolidated to calculate the required number of instances of each VNF and the inter-VNF links of appropriate capacities. The CSPs cloud graph is represented as $G=(C, T)$ where C is the set of available clouds $c_1, c_2 \ldots c_k$ and T is the set of inter-cloud links t_{ij}, where $i \neq j$, $i,j \leq k$. The cloud broker/cloud service provider carries out the task of mapping this chain onto the

available clouds to achieve optimal results for the tenant service provider. In our case, optimality refers to the least cost solution that meets the end-to-end latency threshold requirement. Other QoS parameters like jitter and packet loss can be taken care of in a similar manner. In Fig. 2 we have an example of mapping service chains to multiple clouds with traffic flowing from different areas through different VNFs. The end-to-end latency of the service function chain would depend on the placement of the constituent functions.

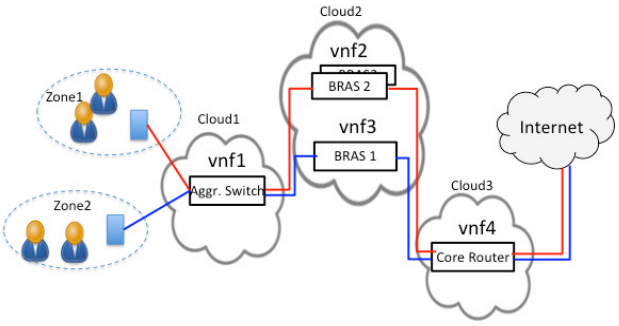

Fig. 2. Mapping service function chain to multi-cloud

IV. The NFV Placement Problem And The Proposed Solution

Placing network functions and programming network flows, in a cost-effective manner, while ensuring acceptable end-to-end delays represents an essential step toward enabling the use of NFV in production environments [21]. From the tenant's point of view the placement problem boils down to placing network functions to meet criteria like cost, jitter, packet loss, latency, throughput in services like voice and video calls, content delivery, broadband services, carrier backbone or a combination of these. The cloud service provider would like to minimize the use of resources while meeting the tenant's requirements. A strategy that reconciles these requirements is the optimal placement strategy. In this work, we discuss a strategy that would primarily optimize cost and at the same time keep end-to-end latency below the threshold prescribed by the carrier. From the discussion, it will become clear that the method would work for other quality of service parameters.

A. *Problem Description*

The cost of placing an SFC is a function of the choice of clouds and the amount of compute, storage and networking resources consumed. Latency depends on a number of factors like the state of compute, networking and storage resources, installed and used capacities of the servers and the links, traffic patterns on the link, the types of functions sharing the servers and distance between clouds. All these factors constitute the state S_t of the multi-cloud at time t. The factors governing the state may change during the course of the system's operation [16]. The amount of latency introduced in a placement by the state of the clouds would, therefore, change over time. Given the state S_t, latency can be calculated using assumptions about the type of traffic, e.g., Poisson and service times and the queuing discipline. Herein lies our *first problem*. The process of creation of virtual resources to host network functions and booting them up takes 15-20 minutes [22]. Loading the network function software for various VNFs and chaining needs additional time. Placement plan based on calculations at time t based on the state S_t is actually carried out at a time $t+1$. This applies to initial placement as well as reconfigurations during operation. Fig. 3 shows a placement request for a 5-VNF service chain over four clouds. At the time of planning, the calculated end-to-end latency is 20 ms. When the actual placement takes place and the VNFs are booted up and chained, the actual latency turns out to be 50ms. This would cause SLA violation right at the inception and trigger reconfiguration of the chain. Reconfiguration may require migration of virtual network functions and re-chaining causing disruption of service. Summing over several service chain instantiations, this can lead to a heavy penalty to be paid by the cloud service providers and a loss of customers and revenue to the communication service provider. This provides the motivation for prediction of the state at $t+1$ so that placement remains consistent with the requirements.

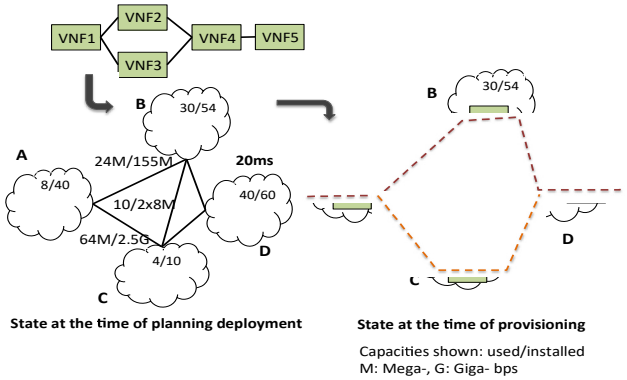

Fig. 3. Demonstrating need for predictive placement

The *second problem* arises from the need to continuously keep latency below the specified threshold during operation. This may require reconfiguration and change of placement (migration). The algorithm should be fast in giving optimum SFC placement decisions for carrying out scaling or migration decisions to dynamically manage the network. This necessitates solving the placement problem fast but the problem is NP hard and ILP based solutions may take hours to converge to the optimum solution [15]. Hence, in many situations they may not be suitable for dynamic placement.

We need fast methods, for example one producing a placement for a 100-function network in the sub-minute region, to make dynamic scaling a reality. Machine learning based predictive methods would fare well in dynamic situation as they work fast and unlike other methods they can capture all the known and unknown interacting parameters. The work that we report in this paper proves this hypothesis.

B. *Solution to the Placement Problem; the COLAP framework*

The proposed COLAP framework has two main components: the first component is a random selection algorithm for fast selection of the least-cost cloud set and the second, an efficient heuristic for lowest cost first (LCF) placement within the selected least-cost cloud set taking into account the predicted latency values at the expected time of placement. The framework provides interfaces for the CSP/Cloud-broker and the tenant TSP. It holds in its databases the cloud configuration data, tenant's SLA (including latency threshold and cost budgets and tariffs. It takes tenant's initial requests and online requests during operation.

1) Optimization by random search: Minimization by random search has been mathematically studied in [23] and has been found to be competitive in many situations. Its application in the random cloud selection algorithm has proved to be quite efficient in our situation. There are two constraints that we are trying to meet simultaneously – cost and latency. We are optimizing cost across all clouds and trying to keep the latency within threshold (L_{th}). The usual method would be to search for m out of total n clouds ($m \leq n$) which give the least total cost and the total latency $\sum_{i=1}^{m} l_i \leq L_{th}$. Searching for m least-cost clouds, with constrained latencies, in an unsorted vector would have worst case time complexity of $O((mn^2-m^2n)$. For a case of selecting 5 clouds out of 100 we end up with about 47,500 iterations. The random selection algorithm (Algorithm 1) converges fast and gives the least cost with a high accuracy.

Algorithm 1: **RANDOM_SELECTION** (**C**, *cv_model, r_clouds*)

```
//C: set of available clouds, cv_model: trained model
init small  //contains the smallest latency cost sum
init lat  // latency
init iter //set iterations large enough for convergence
while (iter)
      init r_clouds //holds final min cost set of clouds
      //find a set of m unique clouds
      while (m_clouds not unique)
            m_clouds ← random set of m clouds from set C
      end while
      //test set r_clouds still has lowest cost and lat≤threshold
      call PREDICT_LATENCY   //uses trained SVR model
          for k=1,m
               lat= lat+lat_k  //initial assessment of total latency
               cost=cost+cost_k
          end for
          if cost<small
              small = cost
              r_clouds←m_clouds
          end if
end while
```

The algorithm selects the desired number of unique clouds by repeated random selection always remembering the lowest cost cloud-set so far that has total latency below the given threshold. When the random selection no longer changes (alternatively, a fixed number of iterations can be used based on empirical studies), the process terminates and the resulting least-cost cloud-set is used for placement of the SFC. The cost includes that of cloud resources and inter-cloud links. The link costs are usually larger and ensure locality of clouds. This total cost and latency are upper bound as not all clouds may be required for placement. Our empirical study validates fast convergence to the global minimum. In one trial a total of fifty experiments were conducted to select the least cost set of five clouds out of ten. Each experiment was performed with 1500 and 1700 iterations. The minimum possible cost was 51 units and latency threshold was set at 150 ms. In the former case, 98% of times the minimum cost of 51 units was reached with latency of 137 ms. In the 1700 iteration case, 100% times the minimum cost clouds were selected with the latency below the threshold. For another trial of 5000 experiments, 50 each with the number of clouds increasing from 10 to 100 and iterations from 500 to 2000, the convergence rate is shown in Fig. 4. Somewhere between 1500 and 2000 iterations the algorithm converges to the minimum cost in 100% cases.

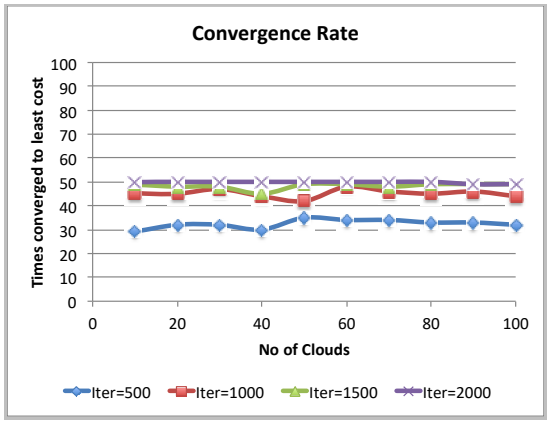

Fig. 4. Number of convergences in 50 experiments

2) Predictive placement: At the heart of our framework is an intelligent latency prediction module that provides inputs for both the initial placement and the scaling functions. The main idea is to use a machine learning based predictive technique to predict latency at the time when the service chain would actually be activated. In the literature, we find use of many supervised machine-learning techniques in cloud computing settings such as: Artificial Neural Networks (ANNs), Bayesian networks, Ensemble classifiers and Support Vector Machines (SVMs). The authors in [24] have observed in their review that SVM is the best technique for classification. Our situation of multi-cloud resources is different from the ones studied.

We worked with a number of methods and find interesting results using Support Vector Regression (SVR), which we will share in this paper. SVR offers the advantage of unique global minimum, shows good generalization properties on real-life test data, and handles non-linear functions. Additionally, we apply cross-validation (both *k*-fold and holdout) to make the method quite useful [25]. As we shall see shortly, appropriate choice of parameters results in predictions with low errors. Examination of literature shows that SVR technique has proved its superiority in problems like resource prediction,

performance modeling, cost-effective storage allocation, and anomaly detection. Some important aspects of SVR are given below for better understanding of its use in COLAP. For a more thorough exposure, readers are referred to [26].

Let $(\boldsymbol{x_i}, y_i)$, $i=1\ldots n$, be sets of data consisting of the predictor vectors $\boldsymbol{x_i}$ and corresponding labels y_i. Each predictor vector $\boldsymbol{x_i}$ consists of values of 'd' selected features that together produce a particular label y_i. Thus, $\boldsymbol{x_i} \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$. In general, vectors in the training set are assumed to be independently and identically distributed from the universal dataset. The goal is to find a function $f(x)$ that represents the labels y_i with a precision of ϵ and is as flat as possible. Since data may be noisy, so finding an exact fit makes approximation too complex and sensitive to errors. It might lead to overfitting. Hence, the use of precision ϵ or the error-insensitive tube within which errors are ignored. In the linear case $f(x)$ can be written as:

$$f(x)=\langle \omega, x\rangle + b \text{ where } \omega \in \mathbb{R}^d, b \in \mathbb{R}^d \quad (1)$$

Here, $\boldsymbol{\omega}$ is the learned weight vector in which all the learning is concentrated and b is the bias. $\langle .,. \rangle$ represents the dot product. Flatness of the function requires that Euclidean norm $\|\omega\|^2$ be minimized. Minimization of $\|\omega\|^2$ is equivalent to maximization of the margin between the classes in the classification case. In the dual form the convex optimization problem can be written as:

Minimize $\frac{1}{2}\|\omega\|^2$
Subject to $y_i - \langle \omega, x_i\rangle - b \le \epsilon$ and $\langle \omega, x_i\rangle + b - y_i \le \epsilon$ (2)

The ½ in the minimization function comes from the width of margin being $2/\|\omega\|$. The value of ϵ can affect the number of support vectors used to construct the regression function. The bigger the ϵ, the fewer support vectors are selected. On the other hand, bigger ϵ values result in more 'flat' estimates. Such a function may, however, not exist, i.e., the convex optimization problem may not be feasible. To make the constraints feasible slack variables ξ and ξ^* are introduced, which allows some points to be on the wrong side of the dividing plane. The optimization problem now becomes:

Minimize $\frac{1}{2}\|\omega\|^2 + C \sum_{i=1}^{n} (\xi + \xi_i)$
Subject to $y_i - \langle \omega, x_i\rangle - b \le \epsilon + \xi$ and $\langle \omega, x_i\rangle + b - y_i \le \epsilon + \xi^*$; $\xi, \xi^* \ge 0$ (3)

The constant C determines the tradeoff between the flatness of f and the amount of error allowed above ϵ. A low C makes the decision surface smooth; a high C aims at classifying all training examples correctly by giving the model freedom to select more samples as support vectors. We choose how significantly the misclassifications should be treated and how large the insensitive loss region should be, by selecting suitable values for the parameters C and ϵ.

ω can be found by writing the Langrangian function and differentiating it with respect to ω. New predictions y' can be found using Langrangian multipliers α_i. We need to minimize the Langrangian functions with respect to ω, b and ξ and maximize with respect to α. This will give values of ω and b. $y' = \sum(\alpha_{i+} + \alpha_{i-}) x_i . x' + b$. A set S of **Support Vectors** x_s can be created by finding the indices i where $0 < \alpha < C$ and ξ_+ or $\xi_- = 0$ as the case may be.

In practice, the data points may not be linearly separable. The data x is projected to a higher dimension using function $\phi(x)$. Then we find a linear discriminant function for transformed data $\phi(x)$. The nonlinear discriminant function is of the form $g(x)=w\phi(x)+w_0$. Poor generalization and computational complexity that may result from projecting data to higher dimensionality involves can be avoided through the use of a kernel function that maps the input feature space of dimension d to a higher dimensional space in which the relation becomes linear. $\boldsymbol{x_i}$ is implicitly replaced by $\Phi(\boldsymbol{x_i})$ by carrying out dot product with $k\langle \boldsymbol{x_i}, \boldsymbol{x}\rangle = \Phi\langle \boldsymbol{x_i}, \boldsymbol{x}\rangle$. In our studies, we have found that performance of RBF is better than the others. This choice is based on the cross validation error in our setting. The RBF kernel has the form given below. Here $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$ are two sample feature vectors and Υ is the parameter that sets the spread of the kernel.

$$K(\boldsymbol{x_i}, \boldsymbol{x_j}) = \exp(-\Upsilon \|x_i - x_j\|) \quad (4)$$

Where Υ is the parameter that sets the spread of the kernel

a) Tuning of parameters: One of the advantages of SVR is that it has a very few parameter to train. The three hyper-parameters that we have focused on are ϵ, C, Υ. Tuning these hyper parameters is one of the main challenges in improving the predictive accuracy. The Υ parameter can be seen as the inverse of the radius of influence of samples selected by the model as support vectors. When Υ is very small, the model is too constrained and cannot capture the complexity or "shape" of the data. If gamma is too large, the radius of the area of influence of the support vectors only includes the support vector itself and no amount of regularization with C will be able to prevent overfitting. The constant C determines the tradeoff between the flatness of 'f' and the amount of error allowed above ϵ. A low C makes the decision surface smooth; a high C aims at classifying all training examples correctly by giving the model freedom to select more samples as support vectors. Most researchers have followed a standard procedure in using a grid search [27] to determine the appropriate values. We used grid search with cross-validation error as the guiding parameter. Both k-fold and 20% holdout methods were used to find the best combination of hyperparameters.

The heuristic for placement works as in Algorithm 2.

Algorithm 2: PLACE_SERVICE_CHAIN (vnf types, demands, traffic)

```
Set up cloud data // all c_k ∈ C and t_kj ∈ T
Set up client data // all v_i ∈ V
Call TRAIN_MODEL (predictors, labels,cv_model)
Latency threshold ← L_th
Cost budget ← C_B
NCloud ← c_N
v_i^c ← demands
n ← length of the service function chain (number of VNFs)
native ← true
if (native==1)
    for v_i, i=1,n      //place as many VNFs as possible in the native cloud
        if c^c_N − c^u_N > v_i^c        // native cloud has unused capacity
            pop v_i
            c^u_N ← c^u_N + v_i^c     // update capacity
```

```
                else
                    break
                end if
            end for
end if
if vnf!=0      // for remaining vnfs
    call RANDOM_SELECTION    //get a set of lowest cost clouds
    sort ascending r_clouds on cost   //r_cloud: set of smallest latency clouds
    while vnf!=0
            place vnfs  //on sorted clouds
            update capacity
            update bandwidth
            update vnfs_placed status
    end while
end if
if all_vnf_placed & latency of chain<L_th & cost of chain<C_B
      output placement details
else
      report failure to place
end if
```

TABLE I. SYMBOL TABLE

Sym-bol	Description	Sym-bol	Description
c_k	Cloud k	c_N	Native cloud
C	Set of all clouds available	v_i^c	Capacity demand for VNF i
t_{kj}	Link from cloud k to j	n	Types of VNFs
T	Set of all inter-cloud links	$c^c{}_N$	Equipped cap of native cloud
v_i	VNF i	$c^u{}_N$	Used cap of native cloud
V	Set of VNFs		
L_{th}	Latency threshold	m	No of clouds selected
C_B	Cost budget	v_i^c	Compute capacity required for VNF i

The placement algorithm uses cloud and tenant data as input. It is also presumed that a separate module for predicting latencies has produced a trained model. The placement normally begins with the native cloud (can be overridden by setting *native = 0*). The algorithm accommodates as many VNFs/services as possible in the native cloud. For the remaining VNFs the SVR module predicts latency of various clouds. This algorithm uses Algorithm 1 to select the set of *m* least cost clouds. The number *m* can be decided to start with enough capacity to place all the VNFs. For the least cost set, the algorithm calculates the assignment of VNFs in the sequence in which they appear in the SFC. The final cost and latency is reported. If the clouds are exhausted and placement is not completed then failure to place is reported. If this case happens frequently then the number *m* needs to be increased.

V. EVALUATION OF THE FRAMEWORK

We have evaluated our framework in two ways: *simulation* using queuing-theoretic model and actual i*mplementation* on Cloudlab. In both the cases, we generate data that is used for training of models using SVR. In this work, we have reported results of the models trained with datasets obtained via simulation.

A. *The experimental set-up*

The experimental set-up consists of the network configuration as shown in Fig. 5. As we shall see in Section V (D), the method scales well for larger number of virtual functions. The traffic entering the aggregation switch (VNF1) splits into two streams, one each going to PE-router1 (VNF2) and PE-router2 (VNF3) based on some policy. Traffic may originate in one of many user clusters. The end-to-end latency of the chain would be greater of the latency given by the two routes VNF1-VNF2-VNF4-VNF5 and VNF1-VNF3-VNF4-VNF5.

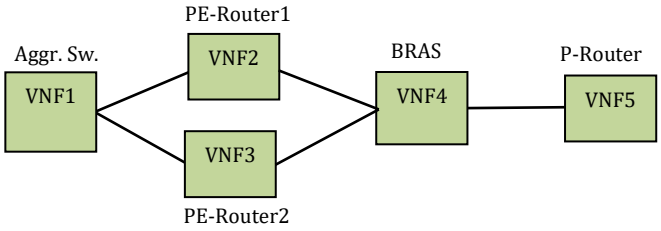

Fig. 5. Experimental service chain configuration

We have used a test configuration of 10 clouds that are fully interconnected. As has been seen in Section IV (B), randomized selection converges fast even for much larger number of clouds. The compute capacities of the VMs hosting VNFs are a single consolidated figure for processor, memory and storage (as in Amazon EC2) and are given in the Table II.

TABLE II. COMPUTER RESOURCE CATEGORIZATION

Integrated capacity	vCPUs	Memory	Storage
1	1	1GB	Flexible
2	2	2GB	Flexible
4	4	4GB	Flexible
6	4	8GB	Flexible
8	8	8GB	Flexible
10	8	16GB	Flexible

The demanded capacities could be fractions of these sizes. The link capacities are chosen from the set {0.016, 0.064, 0.100, 0.155, 0.622, 2.5} representing the capacities in Gbps. The links are presumed to be bi-directional. However, the traffic flow in the experiment is in the direction of ingress at VNF1 to egress at VNF5.

Development of trained prediction models: Selection of the predictor variables forms an important aspect of working with SVR. Too many features make the model complex, increase training time and the test error. Feature selection improves accuracy and speed. We have used cross-validation error as the guiding factor for inclusion of features in our models. The set of variables used for this experiment are given in Table III.

TABLE III. PREDICTOR VARIABLES AND OUTPUT LABEL

	Predictor variables	Label (output)
x_1	Origin cloud compute capacity Installed	y: Latency (ms)
x_2	Destination cloud compute capacity installed	

x_3	Link capacity installed (Gbps)	
x_4	Link capacity used (Gbps)	
x_5	Origin cloud compute capacity used	
x_6	Destination cloud compute capacity used	
x_7	No of user clusters	
x_8	Distance between origin and destination clouds	

The training vector has the form $\boldsymbol{x} = [x_1, x_2, x_3, x_4, x_6, x_7, x_8]^T$ and y is the label. A brief justification for including these features follows. The equipped computer capacities govern the number of VMs created and VNFs instantiated on a server. These VMs cause interference in each other's operations because of shared resources which may lead to delays. Each additional Gbps of equipped capacity does not give the same increase in traffic carrying capacity. The amount of traffic that can actually be carried depends on grade of service required. Traffic depends on the number of clusters and latency depends on traffic requiring this feature to be included. Propagation delay depends on length of the link which is approximated by the distance between the clouds.

B. *Training datasets*

Data plays an important role in building up trained models. The quality and quantity of the datasets will affect the learning and prediction performance. To make the training process credible, we obtained training datasets by two methods – simulation involving node and link queuing delays and test bed implementation on using CloudLab. In simulation, all significant delays: processing delay in the clouds, queueing delay in the virtual machines, propagation delay in the link, queueing delay in the link and transmission delays [28] were accounted for. The network would carry voice, data and video traffic. Some of these appliations are real time and their

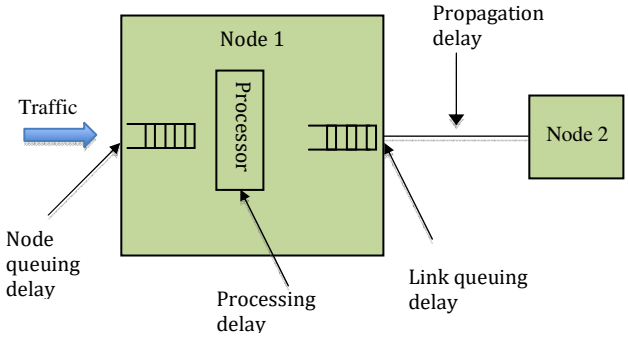

Fig. 6. Major communication delays

packets will go with higher priorioty. Queuing delay is the variable part of the end-to-end delay and depends on the network load and how the traffic behaves. The total time spent in the network by voice and data packets can follow any distribution. It may be presumed that we have a M/G/1 queueing system of infinite capacity with non-preemptive priority [29]. Traffic mixing allows us to use Kleinrock independence approximation to perform queue-wise calculations.

When the traffic exceeds the capacity of a node then there is a *node queuing delay* before the packets can be processed. The *processing delay* includes the time the node spends in error correction and other functions of the node (say flow volume calculation in BRAS) till the time the packet is assigned to an outgoing link queue for transmission. The *link queuing delay* is time between assignment of a packet to a queue and beginning of its transmission. The *transmission delay* is time between the transmission of the first and the last bits of a packet. The *propagation delay* is time between transmission of the last bit at the node and it being received at the next node. Retransmission delays have not been included.

A C++ program was written to generate the dataset using the parameters described above for a special case of M/D/1 queues. The link length feature has been normalized by 500 to keep the numbers comparable with other feature value. The model is trained with latencies depending on the anticipated startup lag. A snapshot of part of one of the training sets is given in Table IV:

TABLE IV. EXTRACT OF A TRAINING DATASET

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	y
1	1	0.3	0.15	0.5	0.5	10	0.1	1.71872
6	2	2.5	1.25	3	1	10	0.4	1.73968
6	2	2.5	1.75	4.2	1.4	10	0.4	1.79711
6	1	2.5	0.5	1.2	0.2	10	0.2	1.80429
2	1	0.064	0.0064	0.2	0.1	10	100.4	7.87455
6	1	0.622	0.311	3	0.5	10	0.6	7.89943
2	6	0.155	0.124	1.6	4.8	10	0.6	7.9167
4	1	0.016	0.0096	2.4	0.6	10	0.6	7.91769
4	6	0.1	0.03	1.2	1.8	10	0.4	4.64659
1	4	0.155	0.0155	0.1	0.4	10	0.2	4.67738
1	6	0.064	0.0256	0.4	2.4	10	0.2	4.70646
2	6	0.155	0.0775	1	3	10	0.4	4.76358
1	1	0.622	0.2488	0.4	0.4	10	0.4	2.2482
2	6	2.5	2	1.6	4.8	10	0.4	2.2689
6	2	0.1	0.04	2.4	0.8	10	0.4	2.34874
1	4	0.155	0.1085	0.7	2.8	10	0.2	2.35271

The features are as described in Section IV (A). The values of x_1, x_2, x_5, and x_6 are integrated capacities as per Table II. The link capacities x_3 and x_4 are in Gbps. The number of clusters x_7 is taken as 10 in the data shown. The inter-cloud distance x_8 is the distance in miles scaled down by a factor of 500.

C. *Simulation results*

The training sets generated by the developed generator program was used to create models in MATLAB R2016a and WEKA. These were tested on a 64 bit machine with i7 Intel quad core processor, L2 Cache (per Core): 256 KB, L3 Cache: 6 MB, and 8 GB RAM. Important outcomes are described below:

1) Tuning of the models: We carried out extensive trials with *k*-fold and 20% holdout cross-validation. It was seen that 20% holdout gives better results in terms of lower errors. Therefore, for arriving at a workable combination of C, Υ and ϵ, we carried out further experiments with 20% holdout cross-

validation. A number of such runs narrowed down the values to $C=1\times10^{-2}$ and $\Upsilon=1$. In both Matlab and Weka, the cross-validation loss with externally tuned parameters was less than that from the system tuned values by as much as an order of 10.

2) SVR Quality of Generalization: The basic idea of using latency prediction to improve placement of virtual functions at the time the service chain will be functional would only work if the predictive model produces good predictions of latency (or any other quality of service parameter that we may choose to work with). The results with the Weka tool show that SVR works quite well on different datasets as can be seen from the training and test root mean square errors (RMSE) (Fig. 7a and 7b). The training dataset had 280 examples and the test set had 56.

```
=== Evaluation on training set ===
=== Summary ===

Correlation coefficient                  0.2137
Mean absolute error                      4.0496
Root mean squared error                  5.2119
Relative absolute error                115.8554 %
Root relative squared error            108.7511 %
Total Number of Instances              280
```

Fig. 7a. SVR training error

```
=== Evaluation on test split ===
=== Summary ===

Correlation coefficient                  0.7304
Mean absolute error                      1.8895
Root mean squared error                  2.5469
Relative absolute error                 63.5334 %
Root relative squared error             71.5849 %
Total Number of Instances               56
```

Fig. 7b. SVR test error

In the Matlab implementation the mean prediction error, the RMSE, and the RMSE test error were of the same order: -5.85, 2.16, and 2.59, respectively.

Low test set RMSE for the used training set show good generalization on unseen test points. For a larger training set the test errors are expected to settle slightly above training errors. This is confirmed with a training set with 2720 examples (Fig. 8a and 8b).

```
=== Evaluation on training set ===
=== Summary ===

Correlation coefficient                  0.7257
Mean absolute error                      1.9259
Root mean squared error                  2.1937
Relative absolute error                101.294  %
Root relative squared error             89.0379 %
Total Number of Instances              2720
```

Fig. 8a. Training error with larger dataset

```
=== Evaluation on test split ===
=== Summary ===

Correlation coefficient                  0.7578
Mean absolute error                      2.4052
Root mean squared error                  2.6691
Relative absolute error                128.401  %
Root relative squared error            112.058  %
Total Number of Instances              544
```

Fig. 8b. Test error with larger dataset

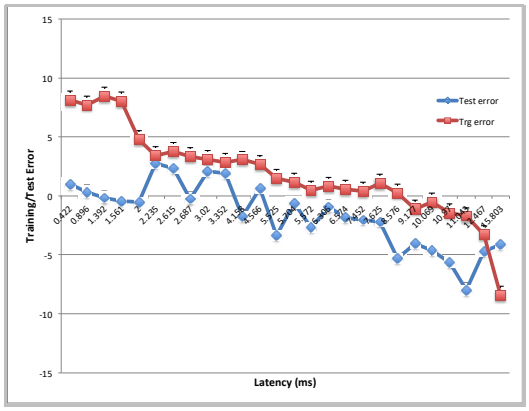

Fig. 9. Training and test errors for the predictive model

A comparative plot of training and test errors obtained through WEKA implementation is given in Fig. 9. The test errors are lower than training errors showing that the predictive model generalizes on unseen data.

D. Placement Speed and Efficiency

It is important for dynamic scaling that the algorithm and heuristic used are able do a large number of placements in a small time. If the algorithm takes a long time in placing the chain or making changes in response to changes in quality of service, or deteriorating performance parameters, then the changes may not be suitable for the situation as it evolves. On the other hand, if maintaining the required performance does not need all the resources that have been contracted then not de-scaling would use up higher amount of resources leading to avoidable expenses. According to various assessments in the literature the run time complexity of training an SVM model is in the range $O(n^2)$ to $O(n^3)$. According to [30] and [31], the complexity is $O(\max(n,d)\ \min(n,d)^2)$ where d is the size of the feature set. If n is much larger than d then it is closer to $O(nd^2)$. However, the time complexity of the search is linear. It took about 1.19 s to train with 2721 examples in Weka and 0.76 s in MATLAB. For speed of placement, we tested with 10 clusters each requesting 10 to 100 SFCs of 5 VNFs each. Thus the number of VNFs varied from 500 to 5000. We see that the algorithm is able to place up to 3000 VNFs less than 1 minute (Fig. 10a).

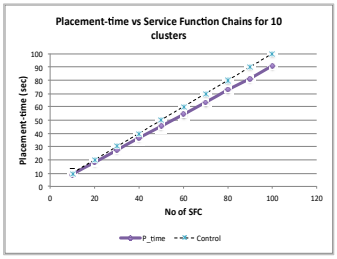

Fig. 10a. Placement time Vs. No. of SFCs

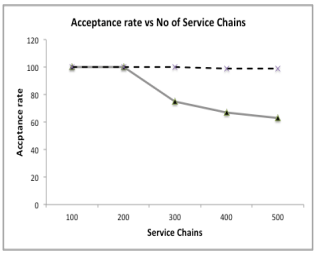

Fig 10b. Acceptance rate Vs No. of SFCs

Finally, any acceptable method should maximize successful tenant request placements. Failure to place service chains would affect tenants quality of service and CSPs revenues. For a medium sized placement request, viz., 100 SFCs or 500 functions, the acceptance rate turns out to be 99-100% (Fig. 10b). As the number of service chains increase, the acceptance rate may fall because of lack of capacity to place the complete service chains. When corrected for capacity the acceptance rate for our algorithm remains above 90% up to the tested configuration of 500 SFC or 2500 VNFs.

VI. SUMMARY AND ONGING WORKS

NFV in multi-cloud environment makes a great business sense both for telecommunication service providers as well as cloud service providers. Meeting network performance parameters is currently difficult in such deployments. Our proposed framework consisting of random cloud selection with SVR based predictive placement allows fast and accurate placement of service function chains optimizing placement cost and meeting the latency threshold.

We are working on enhancing the framework with time adaptive real time-SVR (ART-SVR) for taking care of periodic traffic variatons as well as real-time nature of telecommunications network. The new models are being currently tested both by simulation and on CloudLab.

REFERENCES

[1] NFV Activity Report 2015, https://portal.etsi.org/TBSiteMap/NFV/ActivityReport.aspx

[2] R. Mijumbi, J. Serrat, J-L Gorricho, N. Bouten, F. D. Turck, and S. Davy, "Design and evaluation of algorithms for mapping and scheduling of virtual network functions," IEEE Netsoft, 2015

[3] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "Clickos and the art of network function virtualization," in Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation, 2014.

[4] F. Lopez-Pires and B. Baran, "Virtual machine placement literature review," Polytechnic School, National University of Asuncion, Tech. Rep., 2015, Available: http://arxiv.org/abs/1506.01509

[5] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," Communications Magazine, IEEE, vol. 53, no. 2, pp. 90–97, 2015

[6] "Network Functions Virtualisation (NFV); Infrastructure Overview," ETSI GS NFV-INF 001 V1.1.1, 2015

[7] A. Fischer, J. Botero, M. Till Beck, and H. de Meer, "Virtual network embedding: A survey," in IEEE Communication Surveys, 2013.

[8] K. Li, H. Zheng, and J. Wu, "Migration-based virtual machine Placement in Cloud Systems," IEEE Cloudnet, 2013

[9] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual network functions placement and routing Optimization," IEEE 4th International Conference on Cloud Networoing (CloudNet), 2015

[10] S. Mehraghdam , M. Keller, and H. Karl, "Specifying and placing chains of virtual network functions," IEEE 3rd International Conference on Cloud Networking, CloudNet 2014

[11] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "On orchestrating virtual network functions in NFV," 11th International Conference on Network and Service Management (CNSM), 2015

[12] D. Bhamare, R. Jain, M. Samaka, G. Vaszkun, and A. Erbad, "Multi-Cloud distribution of virtual functions and dynamic service deployment: OpenADN perspective," IEEE International Conference on Cloud Engineering, 2015

[13] S.A. Ajila and A.A. Bankole, "Cloud client prediction models using machine learning techniques," IEEE 37th Annual Computer Software and Applications Conference, 2013

[14] R Mijumbi et al, "Design and evaluation of learning algorithms for dynamic resource management in virtual networks," IEEE Conference on Network Softwarization (NetSoft), 2014

[15] W.A. Rankothge, J. Me, F. Le, A. Russo, and J. Lobo, "Towards making network function virtualization a cloud computing service," IEEE International Symposium on Integrated Network Management (IM), 2015

[16] S. Clayman, E. Maini, and A. Galis, A. Manzalini and N. Mazzocca, "The dynamic placement of virtual networkfFunctions," IEEE Network Operations and Management Symposium (NOMS), 2014

[17] IETF RFC 7498 P. Quinn and T. Nadeau, "Problem statement for service function chaining," IETF, 2015

[18] M. Ghaznavi, N. Shahriar, R. Ahmed, and R. Boutab, "Service function chaining simplified," arXiv:1601.00751, 2016

[19] S. Lee, S. Pack, K. Shin, E. Paik, and R. Browne, "Resource management in service chaining draft-irtf-nfvrg-resource-management-service-chain-03," IETF draft 2016 Lee, Shin

[20] R. Yu, G. Xue, V. T. Kilari, and X. Zhang, "Network function virtualization in the multi-tenant cloud," IEEE Network, 2015

[21] M. C. Luizelli, L. R. Bays, L. S. Buriol M. P. Barcellos, and L. P. Gaspary, "Piecing together the NFV provisioning puzzle: efficient placement and chaining of virtual network functions," IFIP, 2015

[22] Amazon Opworks https://aws.amazon.com/opsworks/, 2016

[23] F. J. Solis and R. J-b. WETS, "Minimization by random search techniques," Mathematics of operations research, 1998

[24] T. Shon and J. Moon, "A hybrid machine learning approach to network anomaly detection," Information Sciences 177 (2007) 3799–3821, Elsevier, 2007

[25] T. Razzaghi, O. Roderick, I. Safro, N. Marko, "Multilevel Weighted Support Vector Machine for Classification on Healthcare Data with Missing Values," PLoS ONE 11(5), 2016

[26] A. J. Smola and B. Scholkopf, "A tutorial on support vector regression," Statistics and Computing, 2004

[27] E. Tuba, L. Mrkela, M. Tuba, "Support vector machine parameter tuning using firefly algorithm," 26th International Conference Radioelektronika, 2016.

[28] D. P. Bertsekas and R. Gallager "Delay models in data networks," Pearson, 1992

[29] V. Gupta, S Dharmaraja, and V Arunachalam,"Stochastic modeling for delay analysis of a VoIP network," Ann Oper Res, Springer Science, 2015

[30] O. Chapelle, "Training a support vector machine in the primal," Neural Computation, 2007.

[31] A. Abdiansah, R. Wardoyo, "Time Complexity Analysis of Support Vector Machines (SVM) in LibSVM," International Journal of Computer Applications, 2015.